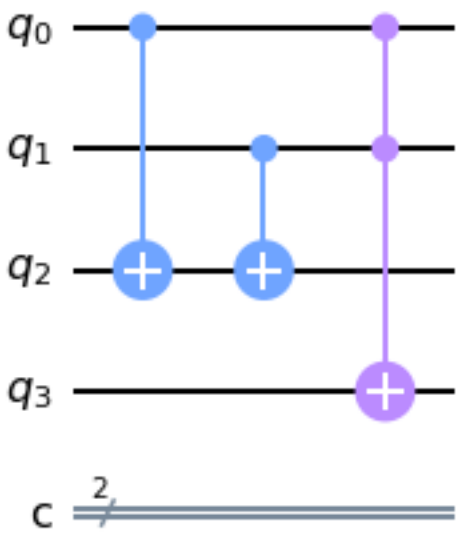
Half_adder circuit

May 25, 2022

[8]:
```
import numpy as np

# Importing standard Qiskit libraries
from qiskit import QuantumCircuit, transpile, Aer, IBMQ
from qiskit.tools.jupyter import *
from qiskit.visualization import *
from ibm_quantum_widgets import *
from qiskit.providers.aer import QasmSimulator

# Loading your IBM Quantum account(s)
provider = IBMQ.load_account()
```

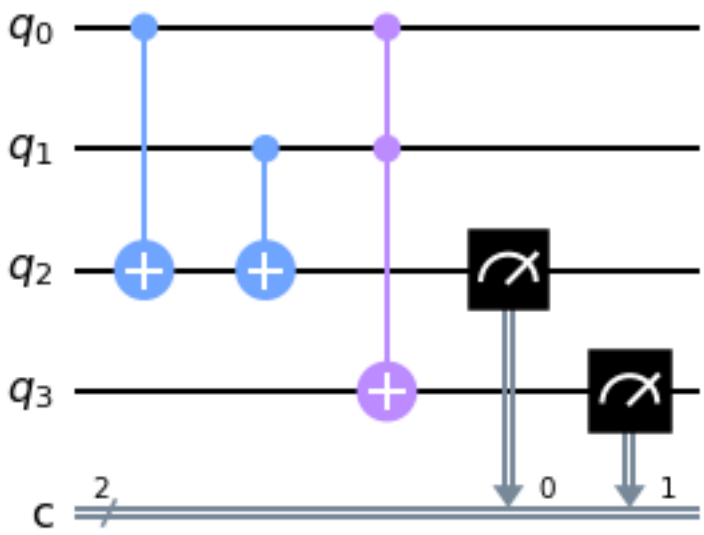
```
ibmqfactory.load_account:WARNING:2022-05-25 11:40:01,657: Credentials are
already in use. The existing account in the session will be replaced.
```

[17]:
```
#Designing the half adder circuit for addition of bits
qc = QuantumCircuit(4, 2) #(4,2) creates 4 qubits and 2 cbits, cbits implies␣
 ↪classical bits
qc.cx(0,2)
qc.cx(1,2)
qc.ccx(0,1,3)
display(qc.draw())      # display a drawing of the circuit

#This circuit adds qubits q0 and q1 and output the result in qubits q2 and q3␣
 ↪which will be measured into cbits c1 and c2 as c1 c2
#E.g. q0 + q1 = 1+0  implies c1 c2 = 01 and q0 + q1 = 1+1  implies c1 c2 = 10
#In classical sense, c2 is the half adder sum, and c1 is the carried over sum
# (Note: measurement is not shown in this design)
```



[10]:
```
#TESTING THE HALF-ADDER CIRCUIT
test_qc = QuantumCircuit(4, 2)
#First, the circuit should encode an input (here '00' which is 0+0)
#By default, qc(0) and qc(1) are in the states zero |0> , so we need do nothing␣
 ↪on them
# Next is to carry out the adder circuit created
test_qc.cx(0,2)
test_qc.cx(1,2)
test_qc.ccx(0,1,3)
# Finally, we will measure the bottom two qubits,q2 and q3 to extract the output
test_qc.measure(2,0) #this measures the 3rd qubit q2, and records it in the 1st␣
 ↪cbit c1
test_qc.measure(3,1) #this measures the 4th qubit q3, and records it in the 2nd␣
 ↪cbit c2
test_qc.draw()
```

[10]:

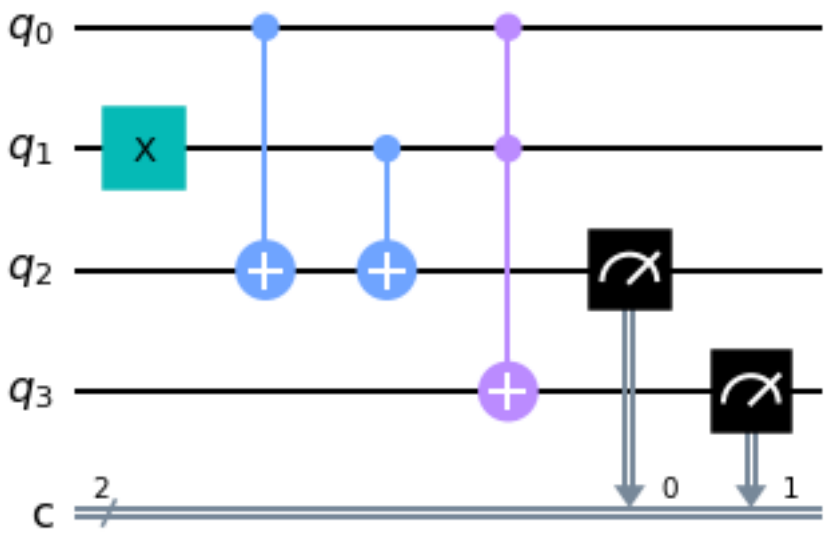


[11]:
```
from qiskit.providers.aer import AerSimulator
sim = AerSimulator()  # make new simulator object
job = sim.run(test_qc)  # runs the experiment
result = job.result()   # get the results
result.get_counts()     # interpret the results as a "counts" dictionary
```

[11]:
```
{'00': 1024}
```

[15]:
```
#In the above we can see that the result '00' was measured 1024 times, and we␣
 ↪didn't measure any other result. This is as expected.
#Now to the next addition, the circuit should encode an input (here '01' which␣
 ↪is 0+1)
test1_qc = QuantumCircuit(4, 2)
#By default, qc(0) and qc(1) are in the states zero|0> , so we need do nothing␣
 ↪on qc(0) but need to flip qc(1)
test1_qc.x(1) #flips qubit q1 from |0> to |1>
# Next is to carry out the adder circuit created
test1_qc.cx(0,2)
test1_qc.cx(1,2)
test1_qc.ccx(0,1,3)
##Finally, we will measure q2 and q3 to extract the output
test1_qc.measure(2,0) #this measures q2 and records it in c1
test1_qc.measure(3,1) #this measures q3 and records it in c2
test1_qc.draw()
```

[15]:

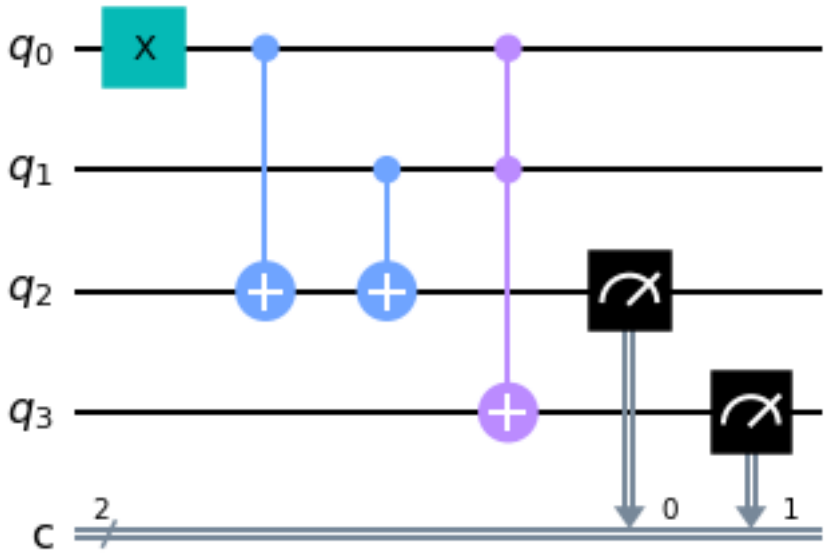


[16]:
```
job1 = sim.run(test1_qc)  # runs the experiment
result1 = job1.result()   # get the results
result1.get_counts()      # interpret the results as a "counts" dictionary
```

[16]:
```
{'01': 1024}
```

[18]:
```
#In the above we can see that the result '01' was measured 1024 times, and we didn't measure any other result. This is as expected.
#Now to the next addition, the circuit should encode an input (here '10' which is 1+0)
test2_qc = QuantumCircuit(4, 2)
#By default, qc(0) and qc(1) are in the states zero|0> , so we need do nothing on qc(1) but need to flip qc(0)
test2_qc.x(0) #flips qubit q0 from |0> to |1>
#Next is to carry out the adder circuit created
test2_qc.cx(0,2)
test2_qc.cx(1,2)
test2_qc.ccx(0,1,3)
#Finally, we will measure q2 and q3 to extract the output
test2_qc.measure(2,0) #this measures q2 and records it in c1
test2_qc.measure(3,1) #this measures q3 and records it in c2
test2_qc.draw()
```

[18]:

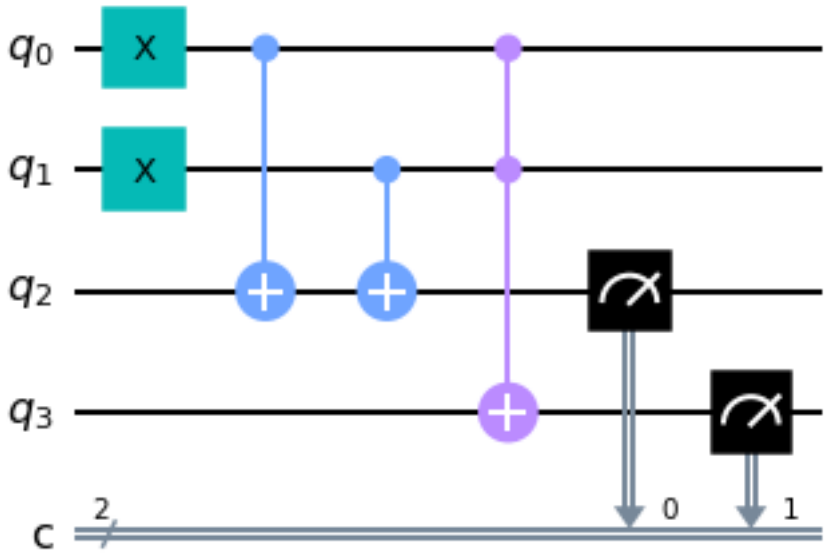


[19]:
```
job2 = sim.run(test2_qc)  # runs the experiment
result2 = job2.result()   # get the results
result2.get_counts()      # interpret the results as a "counts" dictionary
```

[19]:
```
{'01': 1024}
```

[20]:
```
#In the above we again can see that the result '01' was measured 1024 times, and we didn't measure any other result. This is as expected.
#Now to the next addition, the circuit should encode an input (here '11' which is 1+1)
test3_qc = QuantumCircuit(4, 2)
#By default, qc(0) and qc(1) are in the states zero|0> , so we need to flip them
test3_qc.x(0) #flips qubit q0 from |0> to |1>
test3_qc.x(1) #flips q1 from |0> to |1>
#Next is to carry out the adder circuit created
test3_qc.cx(0,2)
test3_qc.cx(1,2)
test3_qc.ccx(0,1,3)
#Finally, we will measure q2 and q3 to extract the output
test3_qc.measure(2,0) #this measures q2 and records it in c1
test3_qc.measure(3,1) #this measures q3 and records it in c2
test3_qc.draw()
```

[20]:



[21]:
```
job3 = sim.run(test3_qc)  # runs the experiment
result3 = job3.result()   # get the results
result3.get_counts()      # interpret the results as a "counts" dictionary
```

[21]:
```
{'10': 1024}
```

[ ]:
```
#In the above we again can see that the result '10' was measured 1024 times,␣
↪and we didn't measure any other result.
#The half adder contains everything needed for addition.
#With the NOT qc.x(arg), CNOT qc.cx(arg), and Toffoli gates qc.ccx(arg), we can␣
↪create programs that add any set of numbers of any size.

#Davikky, 5/25/2022
```